Init

In [234]:

```
try:
    import os
    import glob
    import sys
    import math
    from typing import List, Optional
    from functools import partial
    import itertools
    import copy
except Exception as e:
    print(e)
    print("Some of the libraries needed to run this script were not installed or were not loaded. Please install the libraries befo
re proceeding.")
```

In [235]:

```
sys.path.append(os.environ['DEV_AUTOTS'])
sys.path.append(os.environ['CAPSTONE_PYTHON_SOURCE'])
folder = os.environ['CAPSTONE_DATA']
```

In [236]:

```
try:
    # Data Tables
    import pandas as pd
    import numpy as np

    # Plotting
    import matplotlib.pyplot as plt
    import plotly.offline as py
    from plotly.offline import plot
    py.init_notebook_mode(connected=True)

    # EDA and Feature Engineering
    from scipy.spatial.distance import euclidean, pdist, squareform
    import statsmodels.api as sm

    # Auto Time Series
    import auto_ts as AT

    # Optimizer
    from skopt import gp_minimize
    from skopt.space import Real, Integer
    from skopt.plots import plot_convergence
except Exception as e:
    print(e)
    print("Some of the libraries needed to run this script were not installed or were not loaded. Please install the libraries befo
re proceeding.")
```

In [237]:

```
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [238]:

```
try:
    from ETL.ETL import loadDataset, getTopProducts
    from similarity.similarity import mergeTopSimilar, loadSimilarity
    from charting.charting import surface3DChart
except Exception as e:
    print(e)
    print("Some of the libraries needed to run this script were not installed or were not loaded. Please install the libraries befo
re proceeding.")
```

In [239]:

```
dataRaw= loadDataset(version=4)
```

Prep Data

In [240]:

```python
#Parameters
#ChainMaster = 'SPECS'
#ProdCat='SUP PREM WHISKEY'
TOP_PRODUCTS = 5  # How many products to consider in the category
TOP_SIMILAR = 4  # Get TOP_SIMILAR most similar products

LOG_TRANSFORM = True # Take log of 9L cases to smooth out peaks and valleys
ZERO_ADDER = 0.1

RESAMPLE_FREQ = 'M'

# Pricing changes every 4 weeks
if RESAMPLE_FREQ == 'M':    FORECAST_PERIOD = 1
if RESAMPLE_FREQ == 'W':    FORECAST_PERIOD = 4
if RESAMPLE_FREQ == '2W':   FORECAST_PERIOD = 2

# Seasonal Period
if RESAMPLE_FREQ == 'M':    SEASONAL_PERIOD = 12  # Yearly
if RESAMPLE_FREQ == 'W':    SEASONAL_PERIOD = 13 # Quarterly (we can also take yearly = 52, but SARIMAX becomes too slow)
if RESAMPLE_FREQ == '2W':   SEASONAL_PERIOD = 13 # This becomes problematic --> for quarterly, should we take 6 biweekly periods or 7 bi-weekly periods. Instead I just took half yearly period

print("="*50)
print("Parameters being used...")
print("="*50)
print(f"Resample Frequency = {RESAMPLE_FREQ}")
print(f"Forecast Period = {FORECAST_PERIOD}")
print(f"Seasonal Period = {SEASONAL_PERIOD}")
```

```
==================================================
Parameters being used...
==================================================
Resample Frequency = M
Forecast Period = 1
Seasonal Period = 12
```

Model Flow

Functions

Core Functions

In [241]:

```
COL_TIME = 'WeekDate'
COL_PREDS = ['9L Cases'] #Demand
COL_PRICE= ['Dollar Sales per 9L Case'] #Price

def modelsLoadData(ProductsList,dataRaw,ChainMaster):
    all_data = []

    if(ChainMaster!=''):
        dfSimilarity = loadSimilarity(version=4)
    else:
        dfSimilarity = loadSimilarity(version=4,allCustomers=True)

    for i, Product in enumerate(ProductsList):
        (dataModel,colExog,colEnc,colDec) = mergeTopSimilar(dataRaw, dfSimilarity
                                                            ,ChainMaster=ChainMaster
                                                            ,Product=Product
                                                            ,ProductsList=ProductsList
                                                            ,topn=TOP_SIMILAR
                                                            ,periodCol = COL_TIME
                                                            ,resampleFreq=RESAMPLE_FREQ
                                                            ,encodeCols=True)


        if i == 0: print(f"Decoder: {colDec}")

        print("\n\n")
        print("-"*50)
        print(f"Product: {colDec.get(str(i))}")
        print("-"*50)

        #colExog = colExog + colEndog
        print(f"Exogenous Price Columns: {colExog}")

        allCols=[COL_TIME]+COL_PREDS+ colExog
        data=dataModel[allCols]
        print(f"% of weeks without a purchase: {sum(data['9L Cases'] == 0)/data.shape[0]*100}")
        all_data.append(data)

    all_data_non_transformed =  copy.deepcopy(all_data)

    if LOG_TRANSFORM:
        print("Log Transforming")
        for i in np.arange(len(all_data)):
            all_data_non_transformed[i] = all_data[i].copy(deep=True)
            all_data[i][COL_PREDS] = np.log10(all_data[i][COL_PREDS] + ZERO_ADDER)
```

```python
            print(f"\tProduct: {colDec.get(str(i))}")
    return(all_data,all_data_non_transformed,colExog,colEnc,colDec)

def ModelsWhiteNoise(all_data)           :
    ## WHITE NOISE TEST
    white_noise_all = []
    white_noise_df_all = []
    #check if there are 12, 24, 48 data points
    for i, data in enumerate(all_data):
        lags=[12,24,48]
        lags=[x  for x in lags if x < data.shape[0]]
        white_noise_df = sm.stats.acorr_ljungbox(data[COL_PREDS], lags=lags, return_df=True)
        white_noise_df_all.append(white_noise_df)
        if any(white_noise_df['lb_pvalue'] > 0.05):
            white_noise = True
        else:
            white_noise = False
        white_noise_all.append(white_noise)

        print(white_noise_df)
        print(f"\nIs Data White Noise: {white_noise}")

    return(white_noise_all)

def ModelsTestTrain(all_data,all_data_non_transformed):
    all_train = []
    all_test = []

    all_train_non_transformed = []
    all_test_non_transformed = []

    for i, data in enumerate(all_data):
        train = all_data_non_transformed[i].iloc[:-FORECAST_PERIOD]
        test = all_data_non_transformed[i].iloc[-FORECAST_PERIOD:]
        all_train_non_transformed.append(train)
        all_test_non_transformed.append(test)

        train = data.iloc[:-FORECAST_PERIOD]
        test = data.iloc[-FORECAST_PERIOD:]
        all_train.append(train)
        all_test.append(test)

        print(train.shape,test.shape)
    return(all_train,all_test,all_train_non_transformed,all_test_non_transformed)

def ModelsFit(all_data,all_train,all_test,withSimilar,model_type=['SARIMAX','ML','prophet','auto_SARIMAX']):
    from joblib import Parallel, delayed
```

```
    def modelsFun(i):
        train = all_train[i]
        test = all_test[i]
        import auto_ts as AT
        if(withSimilar==False):
            train = train[train.columns[0:3]] #3rd col has the curr product price
        print(train.columns)

        automl_model = AT.AutoTimeSeries(
            score_type='rmse', forecast_period=FORECAST_PERIOD, # time_interval='Week',
            non_seasonal_pdq=None, seasonality=True, seasonal_period=SEASONAL_PERIOD,
            model_type=model_type,
            verbose=0)

        #colP = COL_PREDS[COL_PREDS in train.columns]
        automl_model.fit(train, COL_TIME, COL_PREDS, cv=10, sep=',') #cv=10
        return(automl_model)

    args = np.arange(len(all_data))

    all_models = Parallel(n_jobs=-1, verbose=1
                          #, backend="threading"
                           , backend="loky"
                         )(
             map(delayed(modelsFun), args))


    return(all_models)

def get_rmse(predictions, targets):
    return np.sqrt(((np.array(predictions) - np.array(targets)) ** 2).mean())

def modelNaive(all_data,all_train,all_test,all_train_non_transformed,season=12,windowLength=8):
    from sktime.forecasting.naive import NaiveForecaster
    import statistics
    from tscv import GapWalkForward # type: ignore
    all_naives=pd.DataFrame(columns=['ID','Best Type','Best RMSE'])
    types=['last','seasonal_last','mean']
    #add window code

    NFOLDS=5
    for i, data in enumerate(all_data):
        yTrain = pd.Series(all_train[i][COL_PREDS[0]])
        yTest = pd.Series(all_test[i][COL_PREDS[0]])
        yTrain = yTrain.append(yTest) # merging as we are gong to do cv
        rmses=[]
        naive_models=[]
        for t in types:
```

```python
            #naive_forecaster = NaiveForecaster(strategy="last")
            cv = GapWalkForward(n_splits=10, gap_size=0, test_size=FORECAST_PERIOD)
            cvRmse=[]
            for fold_number, (train, test) in enumerate(cv.split(yTrain)):
                cv_train = yTrain.iloc[train]
                cv_test = yTrain.iloc[test]

                naive_forecaster = NaiveForecaster(strategy=t,sp=season,window_length=windowLength)
                naive_forecaster.fit(cv_train)
                yPred = naive_forecaster.predict(np.arange(len(cv_test)))
                rmse=get_rmse(yPred, cv_test)
                cvRmse.append(rmse)
            #naive_models.append(naive_forecaster) #last forecaster
            rmses.append(np.mean(cvRmse))
        bestRmse = np.argmin(rmses)
        bestModel = NaiveForecaster(strategy=types[bestRmse],sp=season)
        yTrainNonTrasformed = pd.Series(all_train_non_transformed[i][COL_PREDS[0]])
        bestModel.fit(yTrainNonTrasformed)
        all_naives=all_naives.append(
            {'ID':i
             ,'Best Type': types[bestRmse]
             ,'Best RMSE': rmses[bestRmse]
             ,'Best Naive': bestModel
             ,'All Types': [types]
             ,'All RMSEs': [rmses]
             ,'All Naives':naive_models
            }
            ,ignore_index=True)
    print(all_naives)
    return(all_naives)

def centerLog(text,w,pre='\n',post=''):
    t=int((w-len(text))/2-1)
    return(pre+'='*t+' '+text+' '+'='*(w-len(text)-t-2)+post)

def printLog(main,subs,linesPre=2,linesPost=1):
    import datetime
    if(isinstance(subs,list)== False): subs=[subs]
    maxw=max([len(x) for x in [main] + subs])+10
    print("\n"*linesPre
          +"="*maxw+" ("+str(datetime.datetime.now())+")"
          +centerLog(main,maxw)
          +''.join([centerLog(x,maxw) for x in subs])
          +"\n"+"="*maxw
          +"\n"*linesPost
         )
```

Call Function

In [242]:

```
def runModels(ProductsList,dataRaw,ChainMaster):
    printLog("GET DATA",ChainMaster)
    all_data,all_data_non_transformed,colExog,colEnc,colDec = modelsLoadData(ProductsList,dataRaw,ChainMaster)

    printLog("WHITE NOISE",ChainMaster)
    white_noise = ModelsWhiteNoise(all_data)

    printLog("TEST/TRAIN",ChainMaster)
    all_train, all_test,all_train_non_transformed,all_test_non_transformed = ModelsTestTrain(all_data,all_data_non_transformed)

    all_stats = pd.DataFrame()
    all_stats['Product'] = ProductsList
    all_stats['Chain Master'] = ChainMaster
    all_stats['White Noise'] = white_noise

    printLog("NAIVE",ChainMaster)
    naive = modelNaive(all_data,all_train,all_test,all_data_non_transformed,season=4,windowLength=8)
    all_stats['Naive Best Type'] = [naive.iloc[x]['Best Type'] for x in np.arange(len(all_data)) ]
    all_stats['Naive Best RMSE'] = [naive.iloc[x]['Best RMSE'] for x in np.arange(len(all_data)) ]
    all_stats['Naive Best Model'] = [naive.iloc[x]['Best Naive'] for x in np.arange(len(all_data)) ]

    printLog("Multivar P0",ChainMaster)
    multivarP0 = ModelsFit(all_data,all_train,all_test,withSimilar = False)
    all_stats['P0 Best Model Name'] = [multivarP0[x].get_leaderboard().iloc[0]['name'] for x in np.arange(len(all_data)) ]
    all_stats['P0 Best Model RMSE'] = [multivarP0[x].get_leaderboard().iloc[0]['rmse'] for x in np.arange(len(all_data)) ]
    all_stats['P0 Best Model'] = multivarP0 #[multivarP0[x] for x in np.arange(len(all_data)) ]

    printLog("Multivar P0+Sim",ChainMaster)
    multivarP0Sim = ModelsFit(all_data,all_train,all_test,withSimilar = True )
    all_stats['P0+Sim Best Model Name'] = [multivarP0Sim[x].get_leaderboard().iloc[0]['name'] for x in np.arange(len(all_data)) ]
    all_stats['P0+Sim Best Model RMSE'] = [multivarP0Sim[x].get_leaderboard().iloc[0]['rmse'] for x in np.arange(len(all_data)) ]
    all_stats['P0+Sim Best Model'] = multivarP0Sim #[multivarP0Sim[x] for x in np.arange(len(all_data)) ]

    return(all_stats)
```

Model Setup

In [243]:

```
ChainMasters =  [''] +  dataRaw['Chain Master'].unique().tolist()
ProdCats = dataRaw['Category (CatMan)'].unique().tolist()
display(ChainMasters,ProdCats)
```

```
['', 'THE BARREL HOUSE', 'WESTERN BEV LIQ TX', 'SPECS']
```

```
['ECONOMY VODKA', 'SUP PREM WHISKEY']
```

Testing Models

In [244]:

```
#getting train test
if False:
    ChainMaster=ChainMasters[0]
    ProductsList = getTopProducts(dataRaw, ChainMaster='WESTERN BEV LIQ TX', ProdCat='SUP PREM WHISKEY', topN=TOP_PRODUCTS, timeCol
='WeekDate')
    all_data,all_data_non_transformed,colExog,colEnc,colDec = modelsLoadData(ProductsList,dataRaw,ChainMaster)
    all_train, all_test,all_train_non_transformed,all_test_non_transformed = ModelsTestTrain(all_data,all_data_non_transformed)
```

In [245]:

```
#Fitting model
if False:
    i=1
    withSimilar=False

    train = all_train[i]
    test = all_test[i]
    import auto_ts as AT
    if(withSimilar==False):
        train = train[train.columns[0:3]] #3rd col has the curr product price
    print(train.columns)
    #model_type=['SARIMAX','ML','prophet','auto_SARIMAX']
    model_type=['prophet']
    automl_model = AT.AutoTimeSeries(
        score_type='rmse', forecast_period=FORECAST_PERIOD, # time_interval='Week',
        non_seasonal_pdq=None, seasonality=True, seasonal_period=SEASONAL_PERIOD,
        model_type=model_type,
        verbose=0)

     #colP = COL_PREDS[COL_PREDS in train.columns]
    automl_model.fit(train, COL_TIME, COL_PREDS, cv=1, sep=',') #cv=10
```

In [246]:

```
#prediction
if False:
    display(automl_model.get_leaderboard())
    df=pd.DataFrame({'WeekDate': [pd.to_datetime('2019-12-31')],'0':[266.51]})
    prediction=automl_model.predict(X_exogen = df,forecast_period=1)
    print(prediction)
```

Run

In [247]:

```
full_stats=pd.DataFrame()
ProdCats = ['SUP PREM WHISKEY']
for ProdCat in ProdCats:
    for ChainMaster in ChainMasters:
        printLog("Running ",[ProdCat,ChainMaster])
        ProductsList = getTopProducts(dataRaw, ChainMaster=ChainMaster, ProdCat=ProdCat, topN=TOP_PRODUCTS, timeCol='WeekDate')
        all_stats=runModels(ProductsList,dataRaw,ChainMaster)
        all_stats['Product Category']=ProdCat
        display(all_stats)
        full_stats=full_stats.append(all_stats,ignore_index=True)

printLog("Completed","")
```

```
========================== (2020-08-16 11:25:37.681269)
======== Running  ========
==== SUP PREM WHISKEY ====
============  ============
==========================



================== (2020-08-16 11:25:37.721269)
==== GET DATA ====
========  ========
==================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'JACK DANIELS BLK WHSKY  1.75L', '2': 'JACK DANIELS BLK WHSKY  750
M', '3': 'JACK DANIELS BLK WHSKY SQ 375M', '4': 'GENTLEMAN JACK WHSKY OL 750M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '1', '2', '4', '3']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['1', '0', '2', '4', '3']
% of weeks without a purchase: 1.1904761904761905
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['2', '0', '1', '4', '3']
% of weeks without a purchase: 0.0
resampling to  M
```

```
--------------------------------------------------
Product: JACK DANIELS BLK WHSKY SQ 375M
--------------------------------------------------
Exogenous Price Columns: ['3', '0', '2', '1', '4']
% of weeks without a purchase: 44.047619047619044
resampling to  M



--------------------------------------------------
Product: GENTLEMAN JACK WHSKY OL 750M
--------------------------------------------------
Exogenous Price Columns: ['4', '0', '2', '1', '3']
% of weeks without a purchase: 3.571428571428571
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: JACK DANIELS BLK WHSKY SQ 375M
        Product: GENTLEMAN JACK WHSKY OL 750M


===================== (2020-08-16 11:26:28.913515)
==== WHITE NOISE ====
=========  ==========
=====================

      lb_stat  lb_pvalue
12  17.529696   0.130735
24  31.092108   0.151145
48  54.922995   0.228882

Is Data White Noise: True
       lb_stat     lb_pvalue
12  115.750529  4.333814e-19
24  214.169023  1.845834e-32
48  308.098428  1.176533e-39

Is Data White Noise: False
       lb_stat     lb_pvalue
12   76.707883  1.745018e-11
24  131.122583  9.711501e-17
48  214.957410  5.416649e-23

Is Data White Noise: False
       lb_stat     lb_pvalue
12  236.849902  7.504140e-44
24  313.152234  3.733315e-52
```

```
48  457.992618  2.867236e-68

Is Data White Noise: False
       lb_stat     lb_pvalue
12   49.544022  1.679761e-06
24   97.459679  8.118215e-11
48  147.898014  4.125410e-12

Is Data White Noise: False


==================== (2020-08-16 11:26:28.936516)
==== TEST/TRAIN ====
=========  =========
====================

(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)


=============== (2020-08-16 11:26:28.942514)
==== NAIVE ====
======  =======
===============

  ID Best Type  Best RMSE All Naives  \
0  0      mean   0.043169         []
1  1      last   0.324455         []
2  2      mean   0.427990         []
3  3      mean   0.798321         []
4  4      mean   0.375053         []

                                          All RMSEs  \
0  [[0.07143315389555474, 0.11332894033533458, 0....
1  [[0.3244545696539494, 0.4963154566538675, 0.34...
2  [[0.6704496090247218, 0.719929333484661, 0.427...
3  [[1.158356517777329, 1.1898741527817909, 0.798...
4  [[0.709250053526431, 0.5987185687692762, 0.375...

                      All Types                               Best Naive
0  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
1  [[last, seasonal_last, mean]]                    NaiveForecaster(sp=4)
2  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
3  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
4  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
```

```
===================== (2020-08-16 11:26:29.393469)
==== Multivar P0 ====
=========  ==========
=====================


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  3.5min remaining:  5.2min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  4.1min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.


========================= (2020-08-16 11:30:34.225873)
==== Multivar P0+Sim ====
===========  ============
=========================


[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  3.5min remaining:  5.3min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  4.0min finished
```

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	Naive Best Model	P0 Best Model Name	P0 Best Model RMSE	P0 Best Model	P0+Sim Best Model Name	P0+Sim Best Model RMSE
0	JACK DANIELS BLK WHSKY 1L		True	mean	0.043169	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.027899	<auto_ts.AutoTimeSeries object at 0x00000228CD...	SARIMAX	0.022216
1	JACK DANIELS BLK WHSKY 1.75L		False	last	0.324455	NaiveForecaster(sp=4)	auto_SARIMAX	0.207729	<auto_ts.AutoTimeSeries object at 0x00000228CD...	auto_SARIMAX	0.258516
2	JACK DANIELS BLK WHSKY 750M		False	mean	0.427990	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.278152	<auto_ts.AutoTimeSeries object at 0x00000228D0...	auto_SARIMAX	0.307929
3	JACK DANIELS BLK WHSKY SQ 375M		False	mean	0.798321	NaiveForecaster(sp=4, strategy='mean')	ML	0.625328	<auto_ts.AutoTimeSeries object at 0x000002288B...	ML	0.650942
4	GENTLEMAN JACK WHSKY OL 750M		False	mean	0.375053	NaiveForecaster(sp=4, strategy='mean')	ML	0.253176	<auto_ts.AutoTimeSeries object at 0x00000228CD...	ML	0.256816

```
========================== (2020-08-16 11:34:36.871184)
======== Running  ========
==== SUP PREM WHISKEY ====
==== THE BARREL HOUSE ====
==========================



========================== (2020-08-16 11:34:36.895184)
======== GET DATA ========
==== THE BARREL HOUSE ====
==========================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'JACK DANIELS BLK WHSKY  750M', '2': 'GENTLEMAN JACK WHSKY OL 750M',
'3': 'JACK DANIELS TENN HNY WHSKY  1L', '4': 'GENTLEMAN JACK WHSKY 6PK 1L'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '4', '1', '3', '2']
% of weeks without a purchase: 45.23809523809524
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['1', '0', '2', '4', '3']
% of weeks without a purchase: 59.523809523809526
resampling to  M



--------------------------------------------------
Product: GENTLEMAN JACK WHSKY OL 750M
--------------------------------------------------
Exogenous Price Columns: ['2', '4', '1', '0', '3']
% of weeks without a purchase: 52.38095238095239
resampling to  M
```

```
--------------------------------------------------
Product: JACK DANIELS TENN HNY WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['3', '0', '4', '1', '2']
% of weeks without a purchase: 48.80952380952381
resampling to  M



--------------------------------------------------
Product: GENTLEMAN JACK WHSKY 6PK 1L
--------------------------------------------------
Exogenous Price Columns: ['4', '2', '0', '3', '1']
% of weeks without a purchase: 32.926829268292686
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: GENTLEMAN JACK WHSKY OL 750M
        Product: JACK DANIELS TENN HNY WHSKY  1L
        Product: GENTLEMAN JACK WHSKY 6PK 1L


========================== (2020-08-16 11:35:29.401217)
====== WHITE NOISE =======
==== THE BARREL HOUSE ====
==========================

       lb_stat  lb_pvalue
12   24.932218   0.015147
24   47.266936   0.003107
48  103.010327   0.000007

Is Data White Noise: False
      lb_stat  lb_pvalue
12  22.956927   0.028094
24  26.727008   0.317338
48  60.860438   0.100708

Is Data White Noise: True
       lb_stat  lb_pvalue
12   24.646813   0.016588
24   50.653838   0.001168
48  108.922594   0.000001

Is Data White Noise: False
      lb_stat  lb_pvalue
12  11.113828   0.519190
24  30.880617   0.157255
```

```
48  53.855113   0.260331

Is Data White Noise: True
      lb_stat  lb_pvalue
12   8.641692   0.733192
24  19.315056   0.734985
48  47.523727   0.492264

Is Data White Noise: True


========================== (2020-08-16 11:35:29.418197)
======= TEST/TRAIN =======
==== THE BARREL HOUSE ====
==========================

(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(81, 7) (1, 7)


========================== (2020-08-16 11:35:29.424191)
========= NAIVE ==========
==== THE BARREL HOUSE ====
==========================

  ID      Best Type  Best RMSE All Naives  \
0  0           mean   1.149908         []
1  1           mean   0.959053         []
2  2           last   0.508452         []
3  3  seasonal_last   0.539165         []
4  4           mean   0.611153         []

                                          All RMSEs  \
0  [[1.3378491195367055, 1.5550765479093094, 1.14...
1  [[1.3498152846931788, 1.3498152846931788, 0.95...
2  [[0.5084521358062825, 1.227326606712806, 0.861...
3  [[0.8388226345645782, 0.5391649362857315, 0.68...
4  [[0.7042605406907361, 0.7703384907350979, 0.61...

                      All Types  \
0  [[last, seasonal_last, mean]]
1  [[last, seasonal_last, mean]]
2  [[last, seasonal_last, mean]]
3  [[last, seasonal_last, mean]]
4  [[last, seasonal_last, mean]]
```

```
                                        Best Naive
0            NaiveForecaster(sp=4, strategy='mean')
1            NaiveForecaster(sp=4, strategy='mean')
2                             NaiveForecaster(sp=4)
3   NaiveForecaster(sp=4, strategy='seasonal_last')
4            NaiveForecaster(sp=4, strategy='mean')


========================== (2020-08-16 11:35:29.932221)
====== Multivar P0 =======
==== THE BARREL HOUSE ====
==========================


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  3.0min remaining:  4.6min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  4.0min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.


========================== (2020-08-16 11:39:31.101919)
==== Multivar P0+Sim =====
==== THE BARREL HOUSE ====
==========================


[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  3.1min remaining:  4.6min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  3.8min finished
```

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	Naive Best Model	P0 Best Model Name	P0 Best Model RMSE	P0 Best Model	P0+Sim Best Model Name	P0+S B Mo RM
0	JACK DANIELS BLK WHSKY 1L	THE BARREL HOUSE	False	mean	1.149908	NaiveForecaster(sp=4, strategy='mean')	auto_SARIMAX	0.712256	<auto_ts.AutoTimeSeries object at 0x00000228D5...	SARIMAX	0.7872
1	JACK DANIELS BLK WHSKY 750M	THE BARREL HOUSE	True	mean	0.959053	NaiveForecaster(sp=4, strategy='mean')	ML	0.747042	<auto_ts.AutoTimeSeries object at 0x0000022894...	ML	0.7464
2	GENTLEMAN JACK WHSKY OL 750M	THE BARREL HOUSE	False	last	0.508452	NaiveForecaster(sp=4)	ML	0.615271	<auto_ts.AutoTimeSeries object at 0x0000022894...	ML	0.6160
3	JACK DANIELS TENN HNY WHSKY 1L	THE BARREL HOUSE	True	seasonal_last	0.539165	NaiveForecaster(sp=4, strategy='seasonal_last')	ML	0.141021	<auto_ts.AutoTimeSeries object at 0x0000022885...	ML	0.1390
4	GENTLEMAN JACK WHSKY 6PK 1L	THE BARREL HOUSE	True	mean	0.611153	NaiveForecaster(sp=4, strategy='mean')	ML	0.191544	<auto_ts.AutoTimeSeries object at 0x00000228AD...	ML	0.1886

```
============================ (2020-08-16 11:43:17.731043)
========= Running  =========
===== SUP PREM WHISKEY =====
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-16 11:43:17.756045)
========= GET DATA =========
==== WESTERN BEV LIQ TX ====
============================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1.75L', '1': 'JACK DANIELS BLK WHSKY  750M', '2': 'JACK DANIELS BLK WHSKY  1
L', '3': 'JACK DANIELS BLK WHSKY SQ 375M', '4': 'GENTLEMAN JACK WHSKY OL 750M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['0', '2', '1', '4', '3']
% of weeks without a purchase: 17.5
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['1', '2', '0', '4', '3']
% of weeks without a purchase: 13.414634146341465
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['2', '1', '0', '4', '3']
% of weeks without a purchase: 0.0
resampling to  M
```

```
--------------------------------------------------
Product: JACK DANIELS BLK WHSKY SQ 375M
--------------------------------------------------
Exogenous Price Columns: ['3', '2', '1', '4', '0']
% of weeks without a purchase: 37.096774193548384
resampling to  M



--------------------------------------------------
Product: GENTLEMAN JACK WHSKY OL 750M
--------------------------------------------------
Exogenous Price Columns: ['4', '2', '1', '0', '3']
% of weeks without a purchase: 16.867469879518072
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY SQ 375M
        Product: GENTLEMAN JACK WHSKY OL 750M


============================ (2020-08-16 11:43:58.363631)
======= WHITE NOISE ========
==== WESTERN BEV LIQ TX ====
============================

       lb_stat      lb_pvalue
12  230.533574   1.544168e-42
24  440.855949   2.925831e-78
48  689.590939  1.731957e-114

Is Data White Noise: False
       lb_stat     lb_pvalue
12   77.815839  1.075181e-11
24  136.909527  8.579214e-18
48  201.387337  1.088599e-20

Is Data White Noise: False
      lb_stat     lb_pvalue
12  59.487140  2.799051e-08
24  75.433349  3.193517e-07
48  84.194254  9.632108e-04

Is Data White Noise: False
       lb_stat     lb_pvalue
12  132.198464  2.232893e-22
24  151.760230  1.559044e-20
```

```
48  290.712314  1.863164e-36

Is Data White Noise: False
       lb_stat     lb_pvalue
12   43.376521  1.949365e-05
24  100.761249  2.227445e-11
48  159.701349  6.394533e-14

Is Data White Noise: False


============================ (2020-08-16 11:43:58.376606)
======== TEST/TRAIN ========
==== WESTERN BEV LIQ TX ====
============================

(79, 7) (1, 7)
(81, 7) (1, 7)
(82, 7) (1, 7)
(61, 7) (1, 7)
(82, 7) (1, 7)


============================ (2020-08-16 11:43:58.381608)
========== NAIVE ===========
==== WESTERN BEV LIQ TX ====
============================

  ID      Best Type  Best RMSE All Naives  \
0  0           mean   1.192388         []
1  1           mean   0.706690         []
2  2           mean   0.083096         []
3  3           mean   0.804074         []
4  4  seasonal_last   0.573136         []

                                          All RMSEs  \
0  [[1.209792431243374, 1.8312196882995284, 1.192...
1  [[1.1305923379415006, 1.2624589917738127, 0.70...
2  [[0.12236276035592478, 0.0844886828891572, 0.0...
3  [[1.1553662432502476, 1.1998805840717675, 0.80...
4  [[0.8498829088858146, 0.573135651069236, 0.580...

                        All Types  \
0  [[last, seasonal_last, mean]]
1  [[last, seasonal_last, mean]]
2  [[last, seasonal_last, mean]]
3  [[last, seasonal_last, mean]]
4  [[last, seasonal_last, mean]]
```

```
                                        Best Naive
0           NaiveForecaster(sp=4, strategy='mean')
1           NaiveForecaster(sp=4, strategy='mean')
2           NaiveForecaster(sp=4, strategy='mean')
3           NaiveForecaster(sp=4, strategy='mean')
4  NaiveForecaster(sp=4, strategy='seasonal_last')


============================ (2020-08-16 11:43:58.800635)
======= Multivar P0 ========
==== WESTERN BEV LIQ TX ====
============================


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  2.9min remaining:  4.4min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  3.3min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.


============================ (2020-08-16 11:47:19.799933)
===== Multivar P0+Sim ======
==== WESTERN BEV LIQ TX ====
============================


[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  3.3min remaining:  5.0min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  3.6min finished
```

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	Naive Best Model	P0 Best Model Name	P0 Best Model RMSE	P0 Best Model	P0+Sim Best Model Name
0	JACK DANIELS BLK WHSKY 1.75L	WESTERN BEV LIQ TX	False	mean	1.192388	NaiveForecaster(sp=4, strategy='mean')	auto_SARIMAX	0.713414	<auto_ts.AutoTimeSeries object at 0x00000228B3...	Prophet
1	JACK DANIELS BLK WHSKY 750M	WESTERN BEV LIQ TX	False	mean	0.706690	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.578826	<auto_ts.AutoTimeSeries object at 0x00000228AD...	SARIMAX
2	JACK DANIELS BLK WHSKY 1L	WESTERN BEV LIQ TX	False	mean	0.083096	NaiveForecaster(sp=4, strategy='mean')	ML	0.099956	<auto_ts.AutoTimeSeries object at 0x000002289B...	auto_SARIMAX
3	JACK DANIELS BLK WHSKY SQ 375M	WESTERN BEV LIQ TX	False	mean	0.804074	NaiveForecaster(sp=4, strategy='mean')	auto_SARIMAX	0.643279	<auto_ts.AutoTimeSeries object at 0x00000228CF...	ML
4	GENTLEMAN JACK WHSKY OL 750M	WESTERN BEV LIQ TX	False	seasonal_last	0.573136	NaiveForecaster(sp=4, strategy='seasonal_last')	ML	0.423880	<auto_ts.AutoTimeSeries object at 0x00000228B3...	ML

```
========================== (2020-08-16 11:50:57.639520)
======== Running  ========
==== SUP PREM WHISKEY ====
========= SPECS ==========
==========================



================== (2020-08-16 11:50:57.662518)
==== GET DATA ====
===== SPECS ======
==================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'JACK DANIELS BLK WHSKY  1.75L', '2': 'JACK DANIELS BLK WHSKY  750
M', '3': 'JACK DANIELS TENN HNY WHSKY  1L', '4': 'GENTLEMAN JACK WHSKY OL 750M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '3', '1', '2', '4']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['1', '0', '2', '3', '4']
% of weeks without a purchase: 8.333333333333332
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['2', '0', '1', '3', '4']
% of weeks without a purchase: 2.380952380952381
resampling to  M
```

```
--------------------------------------------------
Product: JACK DANIELS TENN HNY WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['3', '0', '1', '2', '4']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: GENTLEMAN JACK WHSKY OL 750M
--------------------------------------------------
Exogenous Price Columns: ['4', '0', '2', '3', '1']
% of weeks without a purchase: 14.285714285714285
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: JACK DANIELS TENN HNY WHSKY  1L
        Product: GENTLEMAN JACK WHSKY OL 750M


===================== (2020-08-16 11:51:39.091385)
==== WHITE NOISE ====
======= SPECS =======
=====================

      lb_stat  lb_pvalue
12  15.190957   0.231159
24  26.538512   0.326419
48  48.667832   0.445959

Is Data White Noise: True
      lb_stat  lb_pvalue
12  29.265420   0.003598
24  41.411630   0.015005
48  54.239869   0.248702

Is Data White Noise: True
      lb_stat  lb_pvalue
12  26.913026   0.007953
24  38.221972   0.032900
48  54.395929   0.244080

Is Data White Noise: True
      lb_stat  lb_pvalue
12   6.520001   0.887638
24  10.547121   0.991893
```

```
48  32.159954   0.961560

Is Data White Noise: True
      lb_stat  lb_pvalue
12  31.810800   0.001480
24  57.885329   0.000126
48  71.558314   0.015338

Is Data White Noise: False


==================== (2020-08-16 11:51:39.104412)
==== TEST/TRAIN ====
====== SPECS =======
====================

(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)


=============== (2020-08-16 11:51:39.110412)
==== NAIVE ====
==== SPECS ====
===============

  ID Best Type  Best RMSE All Naives  \
0  0      mean   0.060885         []
1  1      mean   0.159196         []
2  2      mean   0.276661         []
3  3      mean   0.075384         []
4  4      mean   0.930727         []

                                           All RMSEs  \
0  [[0.10753730489356994, 0.13940924570407018, 0....
1  [[0.1620938777280673, 0.22871564971857916, 0.1...
2  [[0.4671173585435577, 0.5316177371745061, 0.27...
3  [[0.09595064453644367, 0.0851981539899748, 0.0...
4  [[1.2246381333951109, 1.084772190208572, 0.930...

                      All Types                               Best Naive
0  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
1  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
2  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
3  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
4  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
```

```
===================== (2020-08-16 11:51:39.480412)
==== Multivar P0 ====
======= SPECS =======
=====================


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  2.0min remaining:  3.0min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  2.8min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.


========================= (2020-08-16 11:54:30.435433)
==== Multivar P0+Sim ====
========= SPECS =========
=========================


[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:  2.1min remaining:  3.2min
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  3.7min finished
```

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	Naive Best Model	P0 Best Model Name	P0 Best Model RMSE	P0 Best Model	P0+Sim Best Model Name	P0+Sim Best Model RMSE	P
0	JACK DANIELS BLK WHSKY 1L	SPECS	True	mean	0.060885	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.037469	<auto_ts.AutoTimeSeries object at 0x000002289A...	ML	0.036013	<auto_
1	JACK DANIELS BLK WHSKY 1.75L	SPECS	True	mean	0.159196	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.119452	<auto_ts.AutoTimeSeries object at 0x0000022895...	SARIMAX	0.158028	<auto_
2	JACK DANIELS BLK WHSKY 750M	SPECS	True	mean	0.276661	NaiveForecaster(sp=4, strategy='mean')	auto_SARIMAX	0.237800	<auto_ts.AutoTimeSeries object at 0x000002289A...	SARIMAX	0.256029	<auto_
3	JACK DANIELS TENN HNY WHSKY 1L	SPECS	True	mean	0.075384	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.047690	<auto_ts.AutoTimeSeries object at 0x00000228CE...	SARIMAX	0.051050	<auto_
4	GENTLEMAN JACK WHSKY OL 750M	SPECS	False	mean	0.930727	NaiveForecaster(sp=4, strategy='mean')	ML	0.607617	<auto_ts.AutoTimeSeries object at 0x0000022880...	ML	0.616699	<auto_

```
=================== (2020-08-16 11:58:14.664925)
==== Completed ====
========  =========
===================
```

Print out

In [248]:

```
full_stats[full_stats.columns.difference(['P0 Best Model','P0+Sim Best Model','Naive Best Model'],sort=False)]
```

Out[248]:

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	P0 Best Model Name	P0 Best Model RMSE	P0+Sim Best Model Name	P0+Sim Best Model RMSE	Product Category
0	JACK DANIELS BLK WHSKY 1L		True	mean	0.043169	SARIMAX	0.027899	SARIMAX	0.022216	SUP PREM WHISKEY
1	JACK DANIELS BLK WHSKY 1.75L		False	last	0.324455	auto_SARIMAX	0.207729	auto_SARIMAX	0.258516	SUP PREM WHISKEY
2	JACK DANIELS BLK WHSKY 750M		False	mean	0.427990	SARIMAX	0.278152	auto_SARIMAX	0.307929	SUP PREM WHISKEY
3	JACK DANIELS BLK WHSKY SQ 375M		False	mean	0.798321	ML	0.625328	ML	0.650942	SUP PREM WHISKEY
4	GENTLEMAN JACK WHSKY OL 750M		False	mean	0.375053	ML	0.253176	ML	0.256816	SUP PREM WHISKEY
5	JACK DANIELS BLK WHSKY 1L	THE BARREL HOUSE	False	mean	1.149908	auto_SARIMAX	0.712256	SARIMAX	0.787255	SUP PREM WHISKEY
6	JACK DANIELS BLK WHSKY 750M	THE BARREL HOUSE	True	mean	0.959053	ML	0.747042	ML	0.746450	SUP PREM WHISKEY
7	GENTLEMAN JACK WHSKY OL 750M	THE BARREL HOUSE	False	last	0.508452	ML	0.615271	ML	0.616078	SUP PREM WHISKEY
8	JACK DANIELS TENN HNY WHSKY 1L	THE BARREL HOUSE	True	seasonal_last	0.539165	ML	0.141021	ML	0.139073	SUP PREM WHISKEY
9	GENTLEMAN JACK WHSKY 6PK 1L	THE BARREL HOUSE	True	mean	0.611153	ML	0.191544	ML	0.188641	SUP PREM WHISKEY
10	JACK DANIELS BLK WHSKY 1.75L	WESTERN BEV LIQ TX	False	mean	1.192388	auto_SARIMAX	0.713414	Prophet	0.731563	SUP PREM WHISKEY
11	JACK DANIELS BLK WHSKY 750M	WESTERN BEV LIQ TX	False	mean	0.706690	SARIMAX	0.578826	SARIMAX	0.649111	SUP PREM WHISKEY
12	JACK DANIELS BLK WHSKY 1L	WESTERN BEV LIQ TX	False	mean	0.083096	ML	0.099956	auto_SARIMAX	0.070061	SUP PREM WHISKEY
13	JACK DANIELS BLK WHSKY SQ 375M	WESTERN BEV LIQ TX	False	mean	0.804074	auto_SARIMAX	0.643279	ML	0.613511	SUP PREM WHISKEY
14	GENTLEMAN JACK WHSKY OL 750M	WESTERN BEV LIQ TX	False	seasonal_last	0.573136	ML	0.423880	ML	0.423893	SUP PREM WHISKEY
15	JACK DANIELS BLK WHSKY 1L	SPECS	True	mean	0.060885	SARIMAX	0.037469	ML	0.036013	SUP PREM WHISKEY
16	JACK DANIELS BLK WHSKY 1.75L	SPECS	True	mean	0.159196	SARIMAX	0.119452	SARIMAX	0.158028	SUP PREM WHISKEY

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	P0 Best Model Name	P0 Best Model RMSE	P0+Sim Best Model Name	P0+Sim Best Model RMSE	Product Category
17	JACK DANIELS BLK WHSKY 750M	SPECS	True	mean	0.276661	auto_SARIMAX	0.237800	SARIMAX	0.256029	SUP PREM WHISKEY
18	JACK DANIELS TENN HNY WHSKY 1L	SPECS	True	mean	0.075384	SARIMAX	0.047690	SARIMAX	0.051050	SUP PREM WHISKEY
19	GENTLEMAN JACK WHSKY OL 750M	SPECS	False	mean	0.930727	ML	0.607617	ML	0.616699	SUP PREM WHISKEY

Saving

In [249]:

```
#full_stats.to_pickle('all_Models_stats.pkl')
```

Optimizer

Functions

Optimizer Functions

In [250]:

```python
def complex_objective(x: List
                      , ts_index_name: str
                      , ts_index: List
                      , all_models: List
                      , all_data: List
                      , mask: Optional[List[bool]] = None
                      , verbose: int = 0
                      , return_individual: bool = False
                      , logT = False
                      , P0_only = False
                      #argument for P0 only
                      ):
    """
    :param x A list of product pricing for which the revenue has to be computed
    :type x List
    :param mask: If the customer is not going to purchase a product in a period, we can choose to omit it from the revenue calculat
ion in the optimizer.
                 Default = None (considers all products in revenue calculation)
    :type mask  Optional[List[bool]]

    :param ts_index The index to use for the test data. This is needed for some models (such as ML) that use this to create feature
s
    :type ts_index List

    :param return_individual If True, this returns the individual revenue values as well
                             Used mainly when this function is called standalone. Set of False for optimization
    :type return_individual bool

    :param verbose Level of verbosity (Default: 0). This is set to 1 or 2 (mainly for debug purposes)
    :type verbose int
    """
    if verbose >0: print ("### Prediction Function ###")
    # Create test data from input
    index = [str(i) for i in np.arange(len(x))]
    x_df = pd.DataFrame(x, index = index)
    x_df = x_df.T

    # Set index (important for some models)
    x_df.index = ts_index
    x_df.index.name = ts_index_name

    # If mask is not provided, use all
    if mask is None:
        mask = [False for item in x]
```

```
    if verbose >= 2:
        print(x_df.info())
        print(x_df.columns)

    total_revenue = 0
    revenue = []

    for i in np.arange(len(all_data)):
        if verbose >= 1:
            print("\n" + "-"*50)
            print(f"Product Index: {i}")

        if not mask[i]:
            if P0_only: columns = [all_data[i].columns[-(TOP_SIMILAR+1)]]
            else: columns = all_data[i].columns[-(TOP_SIMILAR+1):].values #columns[-(TOP_SIMILAR+2)] for the P0 only type
            if verbose >= 2:
                print(f"All Columns in Test Data: {columns}")
                print('i:',i)
                print(x_df[columns])
                print("-----------------------------------")

            test_data = x_df[columns]
            prediction = all_models[i].predict(X_exogen = test_data,forecast_period=1) #change this back when Nikhil fixes the auto
TS

            if verbose >= 2: print(f"Prediction Type: {type(prediction)}")
            if verbose >= 1: print(f"Demand Prediction (transformed): {prediction}")

            # If model was created with log transformation
            if logT:
                prediction = 10**prediction
                if verbose >= 1:
                    print("\nDemand Prediction (Original)")
                    print(prediction)

            product_revenue = prediction * x[i]

            # TODO: Clamping - Fix later (this gives an error with pandas. We need to pluck it out as a value)
            # product_revenue = max(product_revenue, 0)  # Clamp at min value of 0 for predictions that are negative

            if verbose >= 1: print(f"Product Revenue: ${round(product_revenue)}")

            if isinstance(product_revenue, pd.Series):
                product_revenue = product_revenue.iloc[0]
            revenue.append(product_revenue)

            # total_revenue = total_revenue + product_revenue
        else:
```

```
            if verbose >= 1: print("This product's revenue was not included since it was not ordered by the customer in this perio
d.")
            product_revenue = 0
            revenue.append(product_revenue)

        if verbose >= 1: print("-"*50 + "\n")

    total_revenue = sum(revenue)

    if verbose >= 1:
        print("\n\n" + "="*50)
        print(f"Total Revenue: ${round(total_revenue)}")
        print("="*50 + "\n\n")
        print ("### Prediction Function END ###")
    if return_individual is True: return -total_revenue, revenue

    return -total_revenue
```

Core Functions

In [251]:

```
def opt_get_mask(all_data,all_test):
    # Did the customer actually want to but products in that period?
    # Only include the revenue in the objective if they actually ordered it
    # This model is not trying to predict if they would purchase a product when they were not going to purchase it earlier.
    # That requires a lot of human psychology and may not be captured in the model

    INCLUDE_MASKING = True

    mask: List[bool] = []
    for index in np.arange(len(all_data)):
        if INCLUDE_MASKING:
            if all_test[index].iloc[0]['9L Cases'] == 0:
                mask.append(True)
            else:
                mask.append(False)
        else:
            mask.append(False)

    print(f"Mask: {mask}")
    return(mask)

def opt_get_space(all_data,MARGIN=0.0):
    MARGIN = 0.0 # How much to go over or under the min and max price respectively during the search for optimial revenue
    space = []

    for index in np.arange(len(all_data)):
        #min_val = all_data[index][str(index)].min()
        min_val = np.percentile(all_data[index][str(index)], 10)
        #max_val = all_data[index][str(index)].max()
        max_val = np.percentile(all_data[index][str(index)], 90)
        min_limit = min_val*(1-MARGIN)
        max_limit = max_val*(1+MARGIN)
        space.append(Real(low=min_limit, high=max_limit, prior='uniform'))

    return(space)

def opt_get_func(all_data,all_models,complex_objective,test_index_name,test_index,mask,verbose=0,P0_only=False):
    # create a new function with mask
    masked_complex_objective = partial(complex_objective, ts_index_name=test_index_name, ts_index=test_index, mask=mask, logT=LOG_T
RANSFORM,verbose=verbose
                                      ,all_models=all_models,all_data=all_data,P0_only=P0_only)
    if False:
        if P0_only:
            print(f"Revenue P0: ${-round(complex_objective([266.51, 195.06, 205.3], ts_index_name=test_index_name, ts_index=test_in
dex, mask=mask,logT=LOG_TRANSFORM,verbose=verbose,all_models=all_models,all_data=all_data,P0_only=True))}")
```

```
        else:
            print(f"Revenue without masking: ${-round(complex_objective([266.51, 195.06, 205.3], ts_index_name=test_index_name, ts_
index=test_index, logT=LOG_TRANSFORM,verbose=verbose,all_models=all_models,all_data=all_data))}")
            print(f"Revenue with masking: ${-round(masked_complex_objective([266.51, 195.06, 205.3],verbose=verbose,all_models=all_
models,all_data=all_data))}")
    return(masked_complex_objective)

def opt_get_data(all_data,all_test_non_transformed):
    total_test_data_revenue = 0
    for index in np.arange(len(all_data)):
        product_price = all_test_non_transformed[index].iloc[0][str(index)]
        product_demand = all_test_non_transformed[index].iloc[0]['9L Cases']
        product_revenue = product_price * product_demand
        print(f"Product {index} Price 9L Case: ${round(product_price,2)} Revenue: ${round(product_revenue)}")
        total_test_data_revenue = total_test_data_revenue + product_revenue

    print(f"Total Revenue: ${round(total_test_data_revenue)}")
    return(total_test_data_revenue)

def opt_naive(all_models,all_test_non_transformed):
    #uses test price and predict demand based on naive model
    product_price=[]
    product_demand=[]
    product_revenue=[]
    for index in np.arange(len(all_models)):
        product_price.append(all_test_non_transformed[index].iloc[0][str(index)])
        product_demand.append(all_models[index].predict([0]).tolist()[0])
        product_revenue.append(product_price[index] * product_demand[index])
    total_revenue = sum(product_revenue)
    return(product_price,product_demand,product_revenue,total_revenue)

def opt_get_chart(all_data,all_models,space,ChainMaster,ProdCat,test_index,test_index_name,verbose=1,STEPS=5,displayPlots=True,save
Path = '3d_charts/'):
    math.ceil(space[0].low)
    math.floor(space[0].high)
    xs = np.arange(math.ceil(space[0].low), math.floor(space[0].high), step=5)
    ys = np.arange(math.ceil(space[1].low), math.floor(space[1].high), step=5)

    allp = [np.arange(math.ceil(space[i].low), math.floor(space[i].high), step=STEPS) for i in np.arange(len(all_data))]

    if verbose >= 1:
        print("-"*100)
        print(f"Price intervals for product 0: {allp[0]}")
        print(f"Price intervals for product 1: {allp[1]}")
        print(f"Price intervals for product 2: {allp[2]}")
        print("-"*100, "\n")
    filenames=[]
    for i in np.arange(len(all_data)):
```

```
        print("\n\n")
        mask_plot = [False if i == j else True for j in np.arange(len(all_data))]
        if verbose >= 1:
            print(f"Product {i} --> Mask: {mask_plot}")

        columns = all_data[i].columns[-(TOP_SIMILAR+1):].values
        if verbose >= 1:
            print(f"Products used in Model: {columns}")

        masked_complex_objective_plot = partial(complex_objective, ts_index_name=test_index_name, ts_index=test_index, mask=mask_pl
ot, logT=LOG_TRANSFORM, verbose=0
                                                ,all_models=all_models,all_data=all_data)

        finalx = []
        finaly = []
        finalrev = []

        xs = allp[int(columns[0])]  # Main Product Price is in xs
        ys = allp[int(columns[1])]  # Exogenous Product Price in in ys

        if verbose >= 1:
            print(f"Price intervals used for X-axis (product {int(columns[0])}): {xs}")
            print(f"Price intervals used for Y-axis (product {int(columns[1])}): {ys}")

        for x, y in itertools.product(xs, ys):
            price_list = [0, 0, 0]

            # Fix price for product 0
            if int(columns[0]) == 0:  # If the main product is product 0
                price_list[0] = x
            elif int(columns[1]) == 0: # If exogenous product is product 0
                price_list[0] = y
            else:
                price_list[0] = 0

            # Fix price for product 1
            if int(columns[0]) == 1:  # If the main product is product 1
                price_list[1] = x
            elif int(columns[1]) == 1: # If exogenous product is product 1
                price_list[1] = y
            else:
                price_list[1] = 0

            # Fix price for product 2
            if int(columns[0]) == 2:  # If the main product is product 2
                price_list[2] = x
            elif int(columns[1]) == 2: # If exogenous product is product 2
                price_list[2] = y
```

```
            else:
                price_list[2] = 0

            rev = -masked_complex_objective_plot(price_list)
            finalx.append(x)
            finaly.append(y)
            finalrev.append(rev)

        fig = surface3DChart(
            x=finalx, y=finaly, z=finalrev,
            title= 'Product ' + columns[0] + ' Revenue',
            xTitle= 'Product ' + columns[0] + ' Price',
            yTitle= 'Product ' + columns[1] + ' Price',
            width=1200,
            height=800
            )

        filename = "".join(ChainMaster.split()) + "_" + "".join(ProdCat.split()) + "_Top" + str(TOP_PRODUCTS) + "_Sim" + str(TOP_SI
MILAR) + \
            "_Log" + str(LOG_TRANSFORM) + "_Add" + str(ZERO_ADDER) + \
            "_Prod" + str(i) + "_Resample" + str(RESAMPLE_FREQ) + "_f" + str(FORECAST_PERIOD) + "_s" + str(SEASONAL_PERIOD) + ".htm
l"
        filenameFull = os.path.join(savePath,filename)
        if verbose >=1: print(filenameFull)
        filenames.append(filenameFull)
        py.plot(fig, filename = filenameFull,auto_open=displayPlots)
    return(filenames)
```

Call Function

In [252]:

```
def runOptimizer(ProductsList,dataRaw,ChainMaster,modelsStats,verbose=0):
    opt_stats = pd.DataFrame()
    numProducts = len(ProductsList)
    opt_stats['Chain Master'] = [ChainMaster] * numProducts
    opt_stats['Product'] = ProductsList


    printLog("GET DATA",ChainMaster)
    all_data,all_data_non_transformed,colExog,colEnc,colDec = modelsLoadData(ProductsList,dataRaw,ChainMaster)

    printLog("TEST/TRAIN",ChainMaster)
    all_train, all_test, all_train_non_transformed, all_test_non_transformed = ModelsTestTrain(all_data,all_data_non_transformed)
    opt_stats['Actual Demand'] = [all_test_non_transformed[x]['9L Cases'].values[0] for x in np.arange(len(all_test_non_transformed
))]
    opt_stats['Actual Price'] = [all_test_non_transformed[x].iloc[0][str(x)] for x in np.arange(len(all_test_non_transformed))]
    opt_stats['Actual Revenue'] =  [opt_stats['Actual Demand'][x] * opt_stats['Actual Price'][x]  for x in np.arange(numProducts)]
    opt_stats['Actual Chain Master Revenue'] =  [sum(opt_stats['Actual Revenue'])] *numProducts

    printLog("NAIVE FORECAST",ChainMaster)
    all_models = modelsStats['Naive Best Model']
    naive_price, naive_demand, naive_revenue ,naive_total_revenue = opt_naive(all_models,all_test_non_transformed) #uses test price
and predict demand based on naive
    opt_stats['Naive Prices'] = naive_price
    opt_stats['Naive Demand'] = naive_demand
    opt_stats['Naive Revenue'] = naive_revenue
    opt_stats['Naive Chain Master Revenue'] = [naive_total_revenue] * numProducts

    printLog("MASK",ChainMaster)
    mask = opt_get_mask(all_data,all_test)
    opt_stats['mask'] = mask

    printLog("SPACE",ChainMaster)
    space = opt_get_space(all_data)
    opt_stats['space'] = space

    printLog("Test Index",ChainMaster)
    test_index_name = 'WeekDate'
    test_index = all_test_non_transformed[0][test_index_name].values
    opt_stats['test_index'] = [test_index] * numProducts# for i in ProductsList]

    #############
    ## P0 Only ##
    if True:
        printLog("GET FUNCTION P0",ChainMaster)
        all_models = modelsStats['P0 Best Model']
```

```
        masked_complex_objective = opt_get_func(all_data,all_models,complex_objective,test_index_name,test_index,mask=mask,verbose=
verbose,P0_only=True)
        opt_stats['masked_complex_objective'] = masked_complex_objective

        printLog("OPTIMIZING P0",ChainMaster)
        res = gp_minimize(masked_complex_objective,
                          space,
                          acq_func="EI",
                          n_calls=200,
                          n_random_starts=20,
                          random_state=42)
        opt_stats['res'] = [res] * numProducts # for i in ProductsList]

        ## GET OUTPUT DATA ##
        printLog("OUTPUT P0",ChainMaster)
        opt_stats['P0 Optimal Price'] = [round(price, 2) for price in res.x]
        opt_stats['P0 Chain Master Revenue'] = round(-res.fun)

        _,all_revenues =  masked_complex_objective(res.x, return_individual=True)
        opt_stats['P0 Demand'] = (np.array(all_revenues) / np.array(opt_stats['P0 Optimal Price'])).tolist()
        opt_stats['P0 Revenue'] = all_revenues

        total_test_data_revenue = opt_get_data(all_data,all_test_non_transformed)
        opt_stats['total_test_data_revenue_P0'] = total_test_data_revenue

    ############
    ## P0+Sim ##
    if True:
        printLog("GET FUNCTION P0+Sim",ChainMaster)
        all_models = modelsStats['P0+Sim Best Model']
        masked_complex_objective = opt_get_func(all_data,all_models,complex_objective,test_index_name,test_index,mask,verbose=verbo
se,P0_only=False)
        opt_stats['masked_complex_objective'] = masked_complex_objective

        printLog("OPTIMIZING P0+Sim",ChainMaster)
        res = gp_minimize(masked_complex_objective,
                          space,
                          acq_func="EI",
                          n_calls=200,
                          n_random_starts=20,
                          random_state=42
                         )
        opt_stats['res'] = [res] * numProducts # for i in ProductsList]

        ## GET OUTPUT DATA ##
        printLog("OUTPUT P0+Sim",ChainMaster)
        opt_stats['P0+Sim Optimal Price'] = [round(price, 2) for price in res.x]
        opt_stats['P0+Sim Chain Master Revenue'] = round(-res.fun)
```

```
        _,all_revenues =  masked_complex_objective(res.x, return_individual=True)
        opt_stats['P0+Sim Demand'] = (np.array(all_revenues) / np.array(opt_stats['P0+Sim Optimal Price'])).tolist()
        opt_stats['P0+Sim Revenue'] = all_revenues

        total_test_data_revenue = opt_get_data(all_data,all_test_non_transformed)
        opt_stats['total_test_data_revenue_P0+Sim'] = total_test_data_revenue

    ############
    # 3D Charts ##
    if False:
        printLog("3D CHARTS",ChainMaster)
        filenames = opt_get_chart(all_data,all_models,space,ChainMaster,ProdCat,test_index,test_index_name,verbose=1,STEPS=5,displa
yPlots=False)
        opt_stats['3d_chart_filenames']  = filenames

    printLog("COMPLETED",ChainMaster)

    return(opt_stats)
```

Loop

In [253]:

```
#reading models data
#full_stats = pd.read_pickle('all_Models_stats.pkl')
#check mask.. change the iteration to 10 random and 20 full
```

In [254]:

```
ChainMasters =  [''] +  dataRaw['Chain Master'].unique().tolist()
ProdCats = dataRaw['Category (CatMan)'].unique().tolist()
display(ChainMasters,ProdCats)
```

```
['', 'THE BARREL HOUSE', 'WESTERN BEV LIQ TX', 'SPECS']
```

```
['ECONOMY VODKA', 'SUP PREM WHISKEY']
```

Testing Models

In [255]:

```
## testing Models Prediction
if False:
    ChainMaster = ChainMasters[2]#Western
    ProdCat = 'SUP PREM WHISKEY'
    modelsStats = full_stats[(full_stats['Chain Master']==ChainMaster) & (full_stats['Product Category']==ProdCat)].reset_index()
    display(modelsStats)
    #display(modelsStats)
    model = modelsStats['P0 Best Model'][1]
    #df=pd.DataFrame({'WeekDate': [pd.to_datetime('2019-12-31')],'0':[266.51],'1':[195.06],'2':[195.06]})
    df=pd.DataFrame({'WeekDate': [pd.to_datetime('2019-12-31')],'1':[266.51]})
    prediction=model.predict(X_exogen = df,forecast_period=1)
    print(prediction)
```

Run

In [256]:

```
full_opt_stats=pd.DataFrame()
ProdCats = ['SUP PREM WHISKEY']
for ProdCat in ProdCats:
    for ChainMaster in ChainMasters:
        modelsStats = full_stats[(full_stats['Chain Master']==ChainMaster) & (full_stats['Product Category']==ProdCat)].reset_index
()

        printLog("Get Top Similar Products",[ProdCat,ChainMaster])
        ProductsList = getTopProducts(dataRaw, ChainMaster=ChainMaster, ProdCat=ProdCat, topN=TOP_PRODUCTS, timeCol='WeekDate')

        printLog("Running Optimizer",[ProdCat,ChainMaster])
        opt_stats=runOptimizer(ProductsList,dataRaw,ChainMaster,modelsStats,verbose=0)

        #display(opt_stats)
        full_opt_stats=full_opt_stats.append(opt_stats,ignore_index=True)

printLog("Completed","")
```

```
================================== (2020-08-16 11:58:16.825610)
==== Get Top Similar Products ====
======== SUP PREM WHISKEY ========
================  ================
==================================



=========================== (2020-08-16 11:58:16.849614)
==== Running Optimizer ====
==== SUP PREM WHISKEY =====
============  ============
===========================



================== (2020-08-16 11:58:16.850580)
==== GET DATA ====
========  ========
==================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'JACK DANIELS BLK WHSKY  1.75L', '2': 'JACK DANIELS BLK WHSKY  750
M', '3': 'JACK DANIELS BLK WHSKY SQ 375M', '4': 'GENTLEMAN JACK WHSKY OL 750M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '1', '2', '4', '3']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['1', '0', '2', '4', '3']
% of weeks without a purchase: 1.1904761904761905
resampling to  M



--------------------------------------------------
```

```
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['2', '0', '1', '4', '3']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY SQ 375M
--------------------------------------------------
Exogenous Price Columns: ['3', '0', '2', '1', '4']
% of weeks without a purchase: 44.047619047619044
resampling to  M



--------------------------------------------------
Product: GENTLEMAN JACK WHSKY OL 750M
--------------------------------------------------
Exogenous Price Columns: ['4', '0', '2', '1', '3']
% of weeks without a purchase: 3.571428571428571
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: JACK DANIELS BLK WHSKY SQ 375M
        Product: GENTLEMAN JACK WHSKY OL 750M


==================== (2020-08-16 11:58:58.349991)
==== TEST/TRAIN ====
=========  =========
====================

(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)


======================== (2020-08-16 11:58:58.357019)
==== NAIVE FORECAST ====
===========  ===========
========================
```

```
============== (2020-08-16 11:58:58.370019)
==== MASK ====
======  ======
==============

Mask: [False, False, False, False, False]


=============== (2020-08-16 11:58:58.371019)
==== SPACE ====
======  =======
===============




==================== (2020-08-16 11:58:58.376019)
==== Test Index ====
=========  =========
====================




========================= (2020-08-16 11:58:58.377019)
==== GET FUNCTION P0 ====
===========  ============
=========================




======================= (2020-08-16 11:58:58.377019)
==== OPTIMIZING P0 ====
==========  ===========
=======================




=================== (2020-08-16 12:02:30.197134)
==== OUTPUT P0 ====
========  =========
===================

Product 0 Price 9L Case: $229.81 Revenue: $135402.0
Product 1 Price 9L Case: $185.65 Revenue: $72331.0
Product 2 Price 9L Case: $222.36 Revenue: $50031.0
Product 3 Price 9L Case: $188.3 Revenue: $23725.0
Product 4 Price 9L Case: $245.31 Revenue: $26984.0
Total Revenue: $308473.0
```

```
============================= (2020-08-16 12:02:30.312134)
==== GET FUNCTION P0+Sim ====
=============  ==============
=============================



=========================== (2020-08-16 12:02:30.312134)
==== OPTIMIZING P0+Sim ====
============  =============
===========================



======================= (2020-08-16 12:06:14.706053)
==== OUTPUT P0+Sim ====
==========  ===========
=======================

Product 0 Price 9L Case: $229.81 Revenue: $135402.0
Product 1 Price 9L Case: $185.65 Revenue: $72331.0
Product 2 Price 9L Case: $222.36 Revenue: $50031.0
Product 3 Price 9L Case: $188.3 Revenue: $23725.0
Product 4 Price 9L Case: $245.31 Revenue: $26984.0
Total Revenue: $308473.0


=================== (2020-08-16 12:06:14.829053)
==== COMPLETED ====
========  =========
===================



================================== (2020-08-16 12:06:14.935050)
==== Get Top Similar Products ====
======== SUP PREM WHISKEY ========
======== THE BARREL HOUSE ========
==================================



=========================== (2020-08-16 12:06:14.954052)
==== Running Optimizer ====
==== SUP PREM WHISKEY =====
==== THE BARREL HOUSE =====
```

```
===========================




========================= (2020-08-16 12:06:14.956052)
======== GET DATA ========
==== THE BARREL HOUSE ====
==========================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'JACK DANIELS BLK WHSKY  750M', '2': 'GENTLEMAN JACK WHSKY OL 750M',
'3': 'JACK DANIELS TENN HNY WHSKY  1L', '4': 'GENTLEMAN JACK WHSKY 6PK 1L'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '4', '1', '3', '2']
% of weeks without a purchase: 45.23809523809524
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['1', '0', '2', '4', '3']
% of weeks without a purchase: 59.523809523809526
resampling to  M



--------------------------------------------------
Product: GENTLEMAN JACK WHSKY OL 750M
--------------------------------------------------
Exogenous Price Columns: ['2', '4', '1', '0', '3']
% of weeks without a purchase: 52.38095238095239
resampling to  M



--------------------------------------------------
Product: JACK DANIELS TENN HNY WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['3', '0', '4', '1', '2']
% of weeks without a purchase: 48.80952380952381
resampling to  M
```

```
--------------------------------------------------
Product: GENTLEMAN JACK WHSKY 6PK 1L
--------------------------------------------------
Exogenous Price Columns: ['4', '2', '0', '3', '1']
% of weeks without a purchase: 32.926829268292686
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: GENTLEMAN JACK WHSKY OL 750M
        Product: JACK DANIELS TENN HNY WHSKY  1L
        Product: GENTLEMAN JACK WHSKY 6PK 1L


========================== (2020-08-16 12:06:55.880797)
======= TEST/TRAIN =======
==== THE BARREL HOUSE ====
==========================

(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(81, 7) (1, 7)


========================== (2020-08-16 12:06:55.886797)
===== NAIVE FORECAST =====
==== THE BARREL HOUSE ====
==========================



========================== (2020-08-16 12:06:55.899797)
========== MASK ==========
==== THE BARREL HOUSE ====
==========================

Mask: [False, False, False, False, False]


========================== (2020-08-16 12:06:55.900797)
========= SPACE ==========
==== THE BARREL HOUSE ====
==========================
```

```
========================== (2020-08-16 12:06:55.906797)
======= Test Index =======
==== THE BARREL HOUSE ====
==========================



========================== (2020-08-16 12:06:55.906797)
==== GET FUNCTION P0 =====
==== THE BARREL HOUSE ====
==========================



========================== (2020-08-16 12:06:55.907797)
===== OPTIMIZING P0 ======
==== THE BARREL HOUSE ====
==========================



========================== (2020-08-16 12:10:18.150371)
======= OUTPUT P0 ========
==== THE BARREL HOUSE ====
==========================

Product 0 Price 9L Case: $239.01 Revenue: $636.0
Product 1 Price 9L Case: $222.36 Revenue: $222.0
Product 2 Price 9L Case: $274.14 Revenue: $3564.0
Product 3 Price 9L Case: $239.01 Revenue: $318.0
Product 4 Price 9L Case: $286.87 Revenue: $1345.0
Total Revenue: $6085.0


============================= (2020-08-16 12:10:18.328370)
==== GET FUNCTION P0+Sim ====
===== THE BARREL HOUSE ======
=============================



=========================== (2020-08-16 12:10:18.328370)
==== OPTIMIZING P0+Sim ====
==== THE BARREL HOUSE =====
===========================
```

```
========================== (2020-08-16 12:13:56.727726)
===== OUTPUT P0+Sim ======
==== THE BARREL HOUSE ====
==========================

Product 0 Price 9L Case: $239.01 Revenue: $636.0
Product 1 Price 9L Case: $222.36 Revenue: $222.0
Product 2 Price 9L Case: $274.14 Revenue: $3564.0
Product 3 Price 9L Case: $239.01 Revenue: $318.0
Product 4 Price 9L Case: $286.87 Revenue: $1345.0
Total Revenue: $6085.0


========================== (2020-08-16 12:13:56.881754)
======= COMPLETED ========
==== THE BARREL HOUSE ====
==========================



================================== (2020-08-16 12:13:56.885725)
==== Get Top Similar Products ====
======== SUP PREM WHISKEY ========
======= WESTERN BEV LIQ TX =======
==================================



============================ (2020-08-16 12:13:56.910727)
==== Running Optimizer =====
===== SUP PREM WHISKEY =====
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-16 12:13:56.912726)
========= GET DATA =========
==== WESTERN BEV LIQ TX ====
============================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1.75L', '1': 'JACK DANIELS BLK WHSKY  750M', '2': 'JACK DANIELS BLK WHSKY  1
L', '3': 'JACK DANIELS BLK WHSKY SQ 375M', '4': 'GENTLEMAN JACK WHSKY OL 750M'}
```

```
--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['0', '2', '1', '4', '3']
% of weeks without a purchase: 17.5
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['1', '2', '0', '4', '3']
% of weeks without a purchase: 13.414634146341465
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['2', '1', '0', '4', '3']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY SQ 375M
--------------------------------------------------
Exogenous Price Columns: ['3', '2', '1', '4', '0']
% of weeks without a purchase: 37.096774193548384
resampling to  M



--------------------------------------------------
Product: GENTLEMAN JACK WHSKY OL 750M
--------------------------------------------------
Exogenous Price Columns: ['4', '2', '1', '0', '3']
% of weeks without a purchase: 16.867469879518072
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY SQ 375M
        Product: GENTLEMAN JACK WHSKY OL 750M
```

```
============================ (2020-08-16 12:14:37.862974)
======== TEST/TRAIN ========
==== WESTERN BEV LIQ TX ====
============================

(79, 7) (1, 7)
(81, 7) (1, 7)
(82, 7) (1, 7)
(61, 7) (1, 7)
(82, 7) (1, 7)


============================ (2020-08-16 12:14:37.869973)
====== NAIVE FORECAST ======
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-16 12:14:37.882973)
=========== MASK ===========
==== WESTERN BEV LIQ TX ====
============================

Mask: [False, False, False, False, False]


============================ (2020-08-16 12:14:37.884974)
========== SPACE ===========
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-16 12:14:37.889973)
======== Test Index ========
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-16 12:14:37.890974)
===== GET FUNCTION P0 ======
==== WESTERN BEV LIQ TX ====
============================
```

```
============================ (2020-08-16 12:14:37.890974)
====== OPTIMIZING P0 =======
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-16 12:18:01.273321)
======== OUTPUT P0 =========
==== WESTERN BEV LIQ TX ====
============================

Product 0 Price 9L Case: $185.59 Revenue: $39085.0
Product 1 Price 9L Case: $222.36 Revenue: $34466.0
Product 2 Price 9L Case: $230.79 Revenue: $25476.0
Product 3 Price 9L Case: $188.3 Revenue: $23725.0
Product 4 Price 9L Case: $241.44 Revenue: $13279.0
Total Revenue: $136032.0


============================= (2020-08-16 12:18:01.364286)
==== GET FUNCTION P0+Sim ====
==== WESTERN BEV LIQ TX =====
=============================



============================ (2020-08-16 12:18:01.365286)
==== OPTIMIZING P0+Sim =====
==== WESTERN BEV LIQ TX ====
============================

Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
```

```
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
```

```
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
```

```
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
```

```
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1


============================ (2020-08-16 12:26:24.633879)
====== OUTPUT P0+Sim =======
==== WESTERN BEV LIQ TX ====
============================
```

```
Building Forecast dataframe. Forecast Period = 1
Product 0 Price 9L Case: $185.59 Revenue: $39085.0
Product 1 Price 9L Case: $222.36 Revenue: $34466.0
Product 2 Price 9L Case: $230.79 Revenue: $25476.0
Product 3 Price 9L Case: $188.3 Revenue: $23725.0
Product 4 Price 9L Case: $241.44 Revenue: $13279.0
Total Revenue: $136032.0


============================ (2020-08-16 12:26:26.118676)
======== COMPLETED =========
==== WESTERN BEV LIQ TX ====
============================



================================== (2020-08-16 12:26:26.124674)
==== Get Top Similar Products ====
======== SUP PREM WHISKEY ========
============ SPECS ==============
==================================



=========================== (2020-08-16 12:26:26.144677)
==== Running Optimizer ====
==== SUP PREM WHISKEY =====
========== SPECS ==========
===========================



================== (2020-08-16 12:26:26.146676)
==== GET DATA ====
===== SPECS ======
==================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'JACK DANIELS BLK WHSKY  1.75L', '2': 'JACK DANIELS BLK WHSKY  750
M', '3': 'JACK DANIELS TENN HNY WHSKY  1L', '4': 'GENTLEMAN JACK WHSKY OL 750M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '3', '1', '2', '4']
```

```
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['1', '0', '2', '3', '4']
% of weeks without a purchase: 8.333333333333332
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['2', '0', '1', '3', '4']
% of weeks without a purchase: 2.380952380952381
resampling to  M



--------------------------------------------------
Product: JACK DANIELS TENN HNY WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['3', '0', '1', '2', '4']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: GENTLEMAN JACK WHSKY OL 750M
--------------------------------------------------
Exogenous Price Columns: ['4', '0', '2', '3', '1']
% of weeks without a purchase: 14.285714285714285
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: JACK DANIELS TENN HNY WHSKY  1L
        Product: GENTLEMAN JACK WHSKY OL 750M


==================== (2020-08-16 12:27:08.168740)
==== TEST/TRAIN ====
====== SPECS =======
```

```
====================

(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)
(83, 7) (1, 7)


======================== (2020-08-16 12:27:08.176768)
==== NAIVE FORECAST ====
======== SPECS =========
========================




=============== (2020-08-16 12:27:08.189768)
==== MASK =====
==== SPECS ====
===============

Mask: [False, False, False, False, False]


=============== (2020-08-16 12:27:08.190768)
==== SPACE ====
==== SPECS ====
===============




==================== (2020-08-16 12:27:08.196771)
==== Test Index ====
====== SPECS =======
====================




========================= (2020-08-16 12:27:08.197769)
==== GET FUNCTION P0 ====
========= SPECS =========
=========================




======================= (2020-08-16 12:27:08.198767)
==== OPTIMIZING P0 ====
======== SPECS ========
```

```
=======================




=================== (2020-08-16 12:30:32.093547)
==== OUTPUT P0 ====
====== SPECS ======
===================

Product 0 Price 9L Case: $229.53 Revenue: $109290.0
Product 1 Price 9L Case: $185.59 Revenue: $32788.0
Product 2 Price 9L Case: $222.36 Revenue: $15343.0
Product 3 Price 9L Case: $229.53 Revenue: $4579.0
Product 4 Price 9L Case: $241.44 Revenue: $10140.0
Total Revenue: $172141.0


============================= (2020-08-16 12:30:32.188549)
==== GET FUNCTION P0+Sim ====
=========== SPECS ===========
=============================




=========================== (2020-08-16 12:30:32.189547)
==== OPTIMIZING P0+Sim ====
========== SPECS ==========
===========================




======================= (2020-08-16 12:34:13.380136)
==== OUTPUT P0+Sim ====
======== SPECS ========
=======================

Product 0 Price 9L Case: $229.53 Revenue: $109290.0
Product 1 Price 9L Case: $185.59 Revenue: $32788.0
Product 2 Price 9L Case: $222.36 Revenue: $15343.0
Product 3 Price 9L Case: $229.53 Revenue: $4579.0
Product 4 Price 9L Case: $241.44 Revenue: $10140.0
Total Revenue: $172141.0


=================== (2020-08-16 12:34:13.464132)
==== COMPLETED ====
====== SPECS ======
===================
```

```
=================== (2020-08-16 12:34:13.467133)
==== Completed ====
========  =========
===================
```

Print out

In [257]:

```
full_opt_stats[['Chain Master','Product'
                ,'Actual Price','Actual Demand','Actual Revenue','Actual Chain Master Revenue'
                ,'Naive Prices','Naive Demand','Naive Revenue','Naive Chain Master Revenue'
                ,'P0 Optimal Price','P0 Demand','P0 Revenue','P0 Chain Master Revenue'
                ,'P0+Sim Optimal Price','P0+Sim Demand','P0+Sim Revenue','P0+Sim Chain Master Revenue'
                ]]
```

Out[257]:

	Chain Master	Product	Actual Price	Actual Demand	Actual Revenue	Actual Chain Master Revenue	Naive Prices	Naive Demand	Naive Revenue	Naive Chain Master Revenue	P0 Optimal Price	P0 Demand
0		JACK DANIELS BLK WHSKY 1L	229.811232	589.19	135402.48	308473.38	229.811232	292.610723	67245.230832	97978.066595	229.38	309.056087
1		JACK DANIELS BLK WHSKY 1.75L	185.650112	389.61	72331.14	308473.38	185.650112	40.950000	7602.372072	97978.066595	183.80	182.760402
2		JACK DANIELS BLK WHSKY 750M	222.360000	225.00	50031.00	308473.38	222.360000	69.590361	15474.112771	97978.066595	219.73	143.646464
3		JACK DANIELS BLK WHSKY SQ 375M	188.295238	126.00	23725.20	308473.38	188.295238	14.013554	2638.685528	97978.066595	192.71	0.493099
4		GENTLEMAN JACK WHSKY OL 750M	245.305091	110.00	26983.56	308473.38	245.305091	20.454795	5017.665391	97978.066595	252.84	1.929937
5	THE BARREL HOUSE	JACK DANIELS BLK WHSKY 1L	239.007519	2.66	635.76	6085.32	239.007519	9.200506	2198.990116	4012.710273	222.49	228.678003
6	THE BARREL HOUSE	JACK DANIELS BLK WHSKY 750M	222.360000	1.00	222.36	6085.32	222.360000	5.218880	1160.470050	4012.710273	237.17	0.125831
7	THE BARREL HOUSE	GENTLEMAN JACK WHSKY OL 750M	274.144615	13.00	3563.88	6085.32	274.144615	0.000000	0.000000	4012.710273	292.22	0.125764
8	THE BARREL HOUSE	JACK DANIELS TENN HNY WHSKY 1L	239.007519	1.33	317.88	6085.32	239.007519	1.330000	317.880000	4012.710273	235.77	2.437597
9	THE BARREL HOUSE	GENTLEMAN JACK WHSKY 6PK 1L	286.874200	4.69	1345.44	6085.32	286.874200	1.169049	335.370107	4012.710273	288.43	1.484818

	Chain Master	Product	Actual Price	Actual Demand	Actual Revenue	Actual Chain Master Revenue	Naive Prices	Naive Demand	Naive Revenue	Naive Chain Master Revenue	P0 Optimal Price	P0 Demand
10	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1.75L	185.589744	210.60	39085.20	136031.88	185.589744	110.560063	20518.813797	53477.231208	191.44	168.625911
11	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 750M	222.360000	155.00	34465.80	136031.88	222.360000	41.041148	9125.909702	53477.231208	217.56	165.061950
12	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1L	230.786122	110.39	25476.48	136031.88	230.786122	66.489207	15344.786307	53477.231208	215.36	58.684028
13	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY SQ 375M	188.295238	126.00	23725.20	136031.88	188.295238	18.149590	3417.481401	53477.231208	222.98	7.122989
14	WESTERN BEV LIQ TX	GENTLEMAN JACK WHSKY OL 750M	241.440000	55.00	13279.20	136031.88	241.440000	21.000000	5070.240000	53477.231208	277.11	6.973551
15	SPECS	JACK DANIELS BLK WHSKY 1L	229.533835	476.14	109290.24	172140.90	229.533835	217.722084	49974.584892	68683.438386	229.53	238.961716
16	SPECS	JACK DANIELS BLK WHSKY 1.75L	185.589744	176.67	32788.14	172140.90	185.589744	44.962771	8344.629157	68683.438386	176.52	41.636904
17	SPECS	JACK DANIELS BLK WHSKY 750M	222.360000	69.00	15342.84	172140.90	222.360000	24.319277	5407.634458	68683.438386	205.46	113.373304
18	SPECS	JACK DANIELS TENN HNY WHSKY 1L	229.533835	19.95	4579.20	172140.90	229.533835	15.903916	3650.486747	68683.438386	226.22	15.541335
19	SPECS	GENTLEMAN JACK WHSKY OL 750M	241.440000	42.00	10140.48	172140.90	241.440000	5.409639	1306.103133	68683.438386	273.82	0.800634

In [258]:

```
full_opt_stats[['Chain Master','Product'
               ,'Actual Price','Actual Demand','Actual Revenue','Actual Chain Master Revenue'
               ,'Naive Prices','Naive Demand','Naive Revenue','Naive Chain Master Revenue'
               ]]
```

Out[258]:

	Chain Master	Product	Actual Price	Actual Demand	Actual Revenue	Actual Chain Master Revenue	Naive Prices	Naive Demand	Naive Revenue	Naive Chain Master Revenue
0		JACK DANIELS BLK WHSKY 1L	229.811232	589.19	135402.48	308473.38	229.811232	292.610723	67245.230832	97978.066595
1		JACK DANIELS BLK WHSKY 1.75L	185.650112	389.61	72331.14	308473.38	185.650112	40.950000	7602.372072	97978.066595
2		JACK DANIELS BLK WHSKY 750M	222.360000	225.00	50031.00	308473.38	222.360000	69.590361	15474.112771	97978.066595
3		JACK DANIELS BLK WHSKY SQ 375M	188.295238	126.00	23725.20	308473.38	188.295238	14.013554	2638.685528	97978.066595
4		GENTLEMAN JACK WHSKY OL 750M	245.305091	110.00	26983.56	308473.38	245.305091	20.454795	5017.665391	97978.066595
5	THE BARREL HOUSE	JACK DANIELS BLK WHSKY 1L	239.007519	2.66	635.76	6085.32	239.007519	9.200506	2198.990116	4012.710273
6	THE BARREL HOUSE	JACK DANIELS BLK WHSKY 750M	222.360000	1.00	222.36	6085.32	222.360000	5.218880	1160.470050	4012.710273
7	THE BARREL HOUSE	GENTLEMAN JACK WHSKY OL 750M	274.144615	13.00	3563.88	6085.32	274.144615	0.000000	0.000000	4012.710273
8	THE BARREL HOUSE	JACK DANIELS TENN HNY WHSKY 1L	239.007519	1.33	317.88	6085.32	239.007519	1.330000	317.880000	4012.710273
9	THE BARREL HOUSE	GENTLEMAN JACK WHSKY 6PK 1L	286.874200	4.69	1345.44	6085.32	286.874200	1.169049	335.370107	4012.710273
10	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1.75L	185.589744	210.60	39085.20	136031.88	185.589744	110.560063	20518.813797	53477.231208
11	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 750M	222.360000	155.00	34465.80	136031.88	222.360000	41.041148	9125.909702	53477.231208
12	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1L	230.786122	110.39	25476.48	136031.88	230.786122	66.489207	15344.786307	53477.231208
13	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY SQ 375M	188.295238	126.00	23725.20	136031.88	188.295238	18.149590	3417.481401	53477.231208
14	WESTERN BEV LIQ TX	GENTLEMAN JACK WHSKY OL 750M	241.440000	55.00	13279.20	136031.88	241.440000	21.000000	5070.240000	53477.231208
15	SPECS	JACK DANIELS BLK WHSKY 1L	229.533835	476.14	109290.24	172140.90	229.533835	217.722084	49974.584892	68683.438386
16	SPECS	JACK DANIELS BLK WHSKY 1.75L	185.589744	176.67	32788.14	172140.90	185.589744	44.962771	8344.629157	68683.438386

	Chain Master	Product	Actual Price	Actual Demand	Actual Revenue	Actual Chain Master Revenue	Naive Prices	Naive Demand	Naive Revenue	Naive Chain Master Revenue
17	SPECS	JACK DANIELS BLK WHSKY 750M	222.360000	69.00	15342.84	172140.90	222.360000	24.319277	5407.634458	68683.438386
18	SPECS	JACK DANIELS TENN HNY WHSKY 1L	229.533835	19.95	4579.20	172140.90	229.533835	15.903916	3650.486747	68683.438386
19	SPECS	GENTLEMAN JACK WHSKY OL 750M	241.440000	42.00	10140.48	172140.90	241.440000	5.409639	1306.103133	68683.438386

In [259]:

```
full_opt_stats[['Chain Master','Product'
               ,'P0 Optimal Price','P0 Demand','P0 Revenue','P0 Chain Master Revenue'
               ,'P0+Sim Optimal Price','P0+Sim Demand','P0+Sim Revenue','P0+Sim Chain Master Revenue'
               ]]
```

Out[259]:

	Chain Master	Product	P0 Optimal Price	P0 Demand	P0 Revenue	P0 Chain Master Revenue	P0+Sim Optimal Price	P0+Sim Demand	P0+Sim Revenue	P0+Sim Chain Master Revenue
0		JACK DANIELS BLK WHSKY 1L	229.38	309.056087	70891.285251	136629.0	229.90	312.739549	71898.822208	191932.0
1		JACK DANIELS BLK WHSKY 1.75L	183.80	182.760402	33591.361834	136629.0	178.04	541.981269	96494.345077	191932.0
2		JACK DANIELS BLK WHSKY 750M	219.73	143.646464	31563.437484	136629.0	208.92	110.929608	23175.413738	191932.0
3		JACK DANIELS BLK WHSKY SQ 375M	192.71	0.493099	95.025183	136629.0	222.99	0.493102	109.956730	191932.0
4		GENTLEMAN JACK WHSKY OL 750M	252.84	1.929937	487.965168	136629.0	282.19	0.896511	252.986525	191932.0
5	THE BARREL HOUSE	JACK DANIELS BLK WHSKY 1L	222.49	228.678003	50878.568817	51948.0	223.18	839.488121	187356.958779	192839.0
6	THE BARREL HOUSE	JACK DANIELS BLK WHSKY 750M	237.17	0.125831	29.843233	51948.0	233.45	19.088156	4456.129927	192839.0
7	THE BARREL HOUSE	GENTLEMAN JACK WHSKY OL 750M	292.22	0.125764	36.750846	51948.0	301.36	0.126122	38.008163	192839.0
8	THE BARREL HOUSE	JACK DANIELS TENN HNY WHSKY 1L	235.77	2.437597	574.712314	51948.0	227.56	3.036507	690.987562	192839.0
9	THE BARREL HOUSE	GENTLEMAN JACK WHSKY 6PK 1L	288.43	1.484818	428.266109	51948.0	297.99	0.995365	296.608707	192839.0
10	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1.75L	191.44	168.625911	32281.744403	84352.0	184.88	1286.789954	237901.726656	266972.0
11	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 750M	217.56	165.061950	35910.877801	84352.0	219.18	11.319866	2481.088336	266972.0
12	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1L	215.36	58.684028	12638.192284	84352.0	214.52	115.008293	24671.578938	266972.0
13	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY SQ 375M	222.98	7.122989	1588.284049	84352.0	181.24	0.515222	93.378813	266972.0
14	WESTERN BEV LIQ TX	GENTLEMAN JACK WHSKY OL 750M	277.11	6.973551	1932.440583	84352.0	261.62	6.972584	1824.167406	266972.0
15	SPECS	JACK DANIELS BLK WHSKY 1L	229.53	238.961716	54848.882611	89227.0	227.74	214.808722	48920.538339	91552.0
16	SPECS	JACK DANIELS BLK WHSKY 1.75L	176.52	41.636904	7349.746373	89227.0	170.16	196.346514	33410.322868	91552.0

	Chain Master	Product	P0 Optimal Price	P0 Demand	P0 Revenue	P0 Chain Master Revenue	P0+Sim Optimal Price	P0+Sim Demand	P0+Sim Revenue	P0+Sim Chain Master Revenue
17	SPECS	JACK DANIELS BLK WHSKY 750M	205.46	113.373304	23293.679091	89227.0	205.73	27.355725	5627.893369	91552.0
18	SPECS	JACK DANIELS TENN HNY WHSKY 1L	226.22	15.541335	3515.760824	89227.0	222.29	15.363519	3415.156625	91552.0
19	SPECS	GENTLEMAN JACK WHSKY OL 750M	273.82	0.800634	219.229673	89227.0	263.58	0.674437	177.768227	91552.0

```
In [ ]:
```